Efficient Ordering and Parameterization of Multi-Linked Negotiation *

Xiaoqin Zhang, Victor Lesser, Sherief Abdallah
Department of Computer Science
University of Massachusetts at Amherst
xqzhang@cs.umass.edu

Abstract

Multi-linked negotiation describes a situation where one agent needs to negotiate with multiple agents about different issues, and the negotiation over one issue influences the negotiations over other issues. In this paper, we present a formalized model of the multi-linked negotiation problem, and describe a search algorithm based on this model for finding the best ordering of negotiation issues and their parameters. Experimental work is presented which shows that this management technique for multi-linked negotiation leads to improved performance.

1 Introduction

Multi-linked negotiation deals with multiple negotiation issues when these issues are interconnected. In a multi-task, resource sharing environment, an agent may need to deal with multiple related negotiation issues including: tasks contracted to other agents, tasks requested by other agents, external resource requirements for local activities and interrelationships among activities distributed among multiple agents.

The potential relationships among these negotiation issues can be classified as two types. One type of relationship is the *directly-linked* relationship: issue B affects issue A directly because issue B is a necessary resource (or a subtask) of issue A. The characteristics (such as cost, duration and quality) of issue B directly affect the characteristics of issue A. For example, as pictured in Figure 1[1], *Computer_Producer_Agent* receives task *Purchase_Computer* from *Consumer_Agent*, and decides if it should accept this task and if it does, what the promised finish time of the task should be. The earlier the task is finished, the higher the reward *Computer_Producer_Agent* can get. In order to accomplish task *Produce_Computer*, *Computer_Producer_Agent* needs to generate an external request for hardware (*Get_Hardware*), and also needs to deliver the computer (*Deliver_Computer*) through a transport agent. The

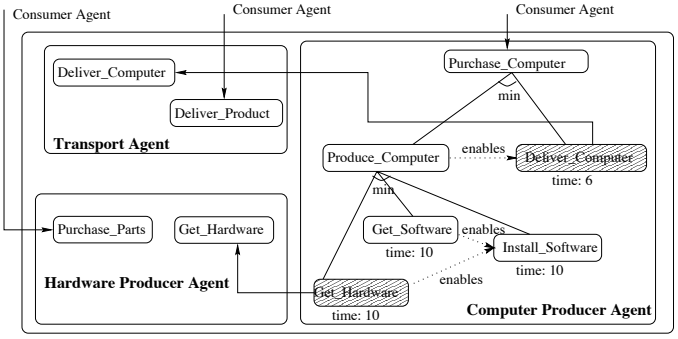

Figure 1: A Supply Chain Scenario

negotiation on the task *Purchase_Computer* is directly linked to the negotiation on the two tasks: *Get_Hardware* and *Deliver_Computer*. If either one of these two tasks fail, the task *Purchase_Computer* can not be accomplished. Furthermore, when and how these two tasks are performed also affect the way the task *Purchase_Computer* is going to be accomplished.

Another type of relationship is the *indirectly-linked* relationship: issue 1 relates to issue 2 because they compete for use of a common resource. For example, as shown in Figure 2, besides the task *Purchase_Computer_A*, *Computer_Producer_Agent* has another contract on task *Purchase_Computer_B*. Because of the limited capability of the *Computer_Producer_Agent*, when task *Purchase_Computer_A* will be performed affects when task *Purchase_Computer_B* can be performed. The negotiation on task *Purchase_Computer_A* and the negotiation on task *Purchase_Computer_B* are *indirectly-linked*.

How can the agent deal with these multiple related negotiation issues? One approach is to deal with these issues independently and ignore their interactions. If these negotiations are performed concurrently, there could be possible conflicts among these issues, hence the agent may not be able to find a combined solution that satisfies all issues without re-negotiation over some already "settled" issues. For example, in Figure 1, *Computer_Producer_Agent* negotiates with *Consumer_Agent* and promises to finish *Purchase_Computer* by time 20. Meanwhile *Computer_Producer_Agent* also negotiates with *Hardware_Producer_Agent* about *Get_Hardware* task and gets a contract that *Get_Hardware* task will be finished at time 20, then *Computer_Producer_Agent* finds it is impossible for *Purchase_Computer* be finished by time 20 given its subtask *Get_Hardware* finished at time 20. To reduce the likelihood that this type of conflict could occur, these negotiations could be performed sequentially. The agent deals with only one negotiation issue at a time, and bases the later negotiations on the results of previous negotiations. However, this sequential process is not a panacea. First of all, the se-

*This material is based upon work supported by the National Science Foundation under Grant No.IIS-9812755, DMI-0122173 and the Air Force Research Laboratory/IFTD and the Defense Advanced Research Projects Agency under Contract F30602-99-2-0525. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. Disclaimer: The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Projects Agency, Air Force Research Laboratory/IFTD, National Science Foundation, or the U.S. Government.

[1]All task plans shown in this paper use the TÆMS language [1], which is also used in our implementation and experiments.

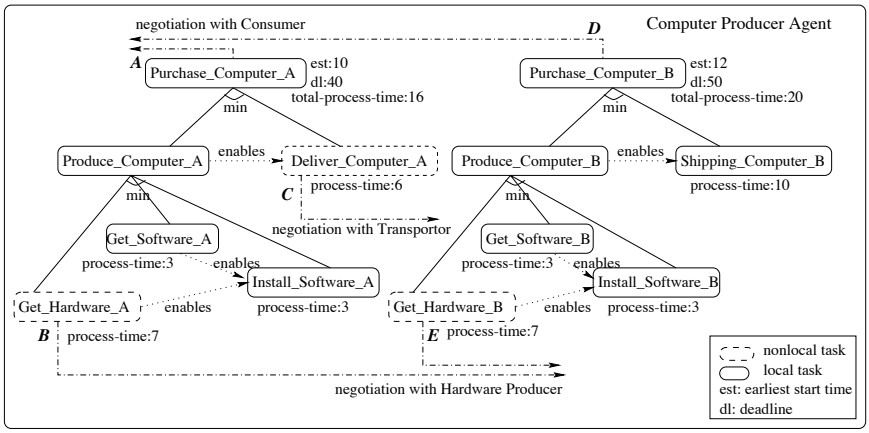

Figure 2: Computer Producer Agent's Tasks

quential negotiation process takes longer, potentially using up valuable time. Secondly there is no guarantee of an optimal solution or even whether any possible solution will be found. The latter problem can occur if the agent does not reason about the ordering of the negotiation issues and just treats them as independent issues, with their ordering random. In this situation, the result from the previous negotiation may make later negotiation issues very difficult or even impossible. For instance, in Figure 1, if *Computer_Producer_Agent* first negotiates about *Purchase_Computer* before starting the negotiations on *Get_Hardware* and *Deliver_Computer*, and the promised finish time of *Purchase_Computer* results in tight constraints on the negotiation on *Get_Hardware* and *Deliver_Computer*, then these negotiations may fail and the commitment on *Purchase_Computer* has to be decommitted.

These previous examples show us how important it is for an agent to reason about the interactions among different negotiation issues and manage them from a more global perspective. If done effectively, this permits the agent to minimize the possibility of conflicts among these different negotiation issues, and achieve better performance. Previously, Zhang and Lesser[3] have presented a partial order schedule as a basic reasoning tool for the agent to deal with multi-linked negotiation. It can be used to reason about the influence of a commitment of one issue on other negotiation issues concerned about time. However, that work does not provide a general solution for how an agent should perform multi-linked negotiations.

In this paper, we introduce a decision-making process that enables an agent to manage the multi-linked negotiation issues and choose the best negotiation approach based on the knowledge about each negotiation issue and the interrelationships among them. The remainder of this paper is structured in the following manner. Section 2 presents a definition of the multi-linked negotiation problem and the algorithms. Section 3 details how the algorithms and the decision-making process work for a multi-linked negotiation problem using an example. Section 4 reports the experimental work to evaluate this approach, and Section 5 presents the major conclusions and the areas of further work.

2 Multi-Linked Negotiation

2.1 Definition of the problem

A multi-linked negotiation problem occurs when an agent has multiple negotiation issues that are related to each other.

Definition 2.1 *A* **multi-linked negotiation problem** *is defined as an undirected graph (more specifically, a forest as a set of rooted trees): $\mathcal{M} = (V, E)$, where $V = \{v\}$ is a finite set of negotiation issues, and $E = \{(u, v)\}$ is a set of binary relations on V. $(u, v) \in E$ denotes that the negotiation on u and the negotiation on v are directly-linked. Each negotiation issue $v_i (v_i \in V)$ is associated with a set of attributes $\mathcal{A}_i = \{a_{ij}\}$. Each attribute a_{ij} either already has been determined (already has a value) or needs to be decided (to be assigned to a value). The relationships among the negotiation issues are described by a forest, a set of rooted trees $\{T_i\}$. There is a relation operator associated with every non-leaf issue v on the tree (denoted as $\rho(v)$), which describes the relationship between this issue and its children. This relation operator has two possible values: AND and OR.*

A negotiation issue v may be a task to be allocated or a resource to be required through negotiation. From an agent's viewpoint, there are two types of negotiation issues:

1. **Incoming negotiation issue**: A task proposed by another agent. For example, issue A (*Purchase_Computer_A*) and D (*Purchase_Computer_B*) in Figure 2 are incoming negotiation issues for *Computer_Producer_Agent*.
2. **Outgoing negotiation issue**: A task needed to be subcontracted to another agent, or a resource requested for a local task. For example, issue B (*Get_Hardware_A*), C (*Deliver_Computer_A*) and E (*Get_Hardware_B*) in Figure 2 are outgoing negotiation issues for *Computer_Producer_Agent*.

Definition 2.2 *A negotiation issue v is* **successful** *(denoted as $S(v)$) if and only if a commitment has been established and confirmed for this issue by all agents involved in this negotiation.*

Definition 2.3 *A leaf node v is* **task-level successful** *(denoted as $TS(v)$) if and only if v is* **successful** *($S(v) = true$); A non-leaf node v is* **task-level successful** *(denoted as $TS(v)$) if and only if the following conditions are fulfilled:*

- *v is* **successful** *($S(v) = true$);*
- *all its children are* **task-level successful** *if $\rho(v) = AND$; or at least one of its children is* **task-level successful**, *if $\rho(v) = OR$.*

For each negotiation issue v_i, $p_s(v_i)$ denotes the **success probability**, the probability that v_i is **successful** ($p(S(v_i) = true)$), there is a function ζ_i mapping the values of the attribute $a_{ij}, j = 1, 2, ..., k$, to $p_s(v_i)$: $p_s(v_i) = \zeta_i(a_{i1}, a_{i2}, ..., a_{ik})$

$\beta(v_i)$ denotes the **decommitment penalty** [2] of v_i. If v_i is **successful** ($S(v_i) = true$), but v_k isn't **task-level successful**

($TS(v_k) = false$), where v_k is the root of the tree that v_i belongs to (denotes as $v_k = root(v_i)$), the utility of the agent decreases by the amount of $\beta(v_i)$ (it represents the penalty paid to the other agent which is involved in the negotiation of issue v_i.) There is a function η_i mapping the values of the attribute $a_{ij}, j = 1, 2, ..., m$, to $\beta(v_i)$: $\beta(v_i) = \eta_i(a_{i1}, a_{i2}, ..., a_{im})$

If v_i is a root of a tree, then $\gamma(v_i)$ denotes the **task-level successful reward** of v_i. The agent's utility increases by the amount of $\gamma(v_i)$ when v_i is **task-level successful**. There is a function θ_i mapping the values of the attribute a_{ij}, $j = 1, 2, ..., m$, to $\gamma(v_i)$: $\gamma(v_i) = \theta_i(a_{i1}, a_{i2}, ..., a_{im})$

The attributes of a negotiation issue are domain dependent. They are specified according to the application domain. The above functions ζ_i, η_i and θ_i are also defined according to the application domain. However, there are some common attributes introduced here:

1. negotiation duration ($\delta(v_i)$): the time needed for the negotiation on v_i to get a result, either success or failure.
2. negotiation start time ($\alpha(v_i)$): the start time of the negotiation on v_i. $\alpha(v_i)$ is an attribute that needs to be decided by the agent.
3. negotiation deadline ($\epsilon(v_i)$): the negotiation on v_j needs to be finished before this deadline $\epsilon(v_i)$. The negotiation is no longer valid after time $\epsilon(v_i)$, which is the same as a failure outcome of this negotiation. For example, if a task v_i is proposed for negotiation, the contractee agent needs to reply before time $\epsilon(v_i)$. Otherwise, this proposed task contract is no longer valid and the contractor agent would think the contractee agent is not interested in this task. Furthermore, even if the agent starts the negotiation before $\epsilon(v_i)$, it is not necessarily true that all times before $\epsilon(v_i)$ are equally good. Usually, an earlier started negotiation has a better chance to succeed for two reasons: the other party considers this issue before other later arriving issues, and this issue has a bigger time range for negotiation. This relationship is described by the function ζ_i that takes $\alpha(v_i)$ as one of its parameters.

2.2 Description of the solution

Given this multi-linked negotiation problem $\mathcal{M} = (V, E)$, an agent needs to make a decision about how the negotiation on these issues should be performed. The decision concerns the following two questions:

1. How are the negotiations on each issue ordered? Should the negotiation on each issue be parallel or sequential? If sequential, in what order? Or should some of them be performed in parallel, while others be sequenced?
2. What values should be assigned to the attributes of each issue that needs to be decided in negotiation?

Definition 2.4 *A* **negotiation ordering** ϕ *is a directed acyclic graph (DAG),* $\phi = (V, E_\phi)$*. If* $(v_i, v_j) \in E_\phi$*, then the negotiation on* v_j *can only start after the negotiation on* v_i *has been completed.*

Definition 2.5 *A negotiation schedule* $\mathcal{NS}(\phi)$ *contains a set of negotiation issues* $\{v_i\}$*. Each issue* v_i *has its negotiation start time* $\alpha(v_i)_\phi$ *and its negotiation finish time* $\varepsilon(v_i)_\phi$ *calculated based on its negotiation duration* $\delta(v_i)$*.*

Using the topological sorting algorithm, a negotiation schedule $\mathcal{NS}(\phi)$ can be generated from a negotiation ordering ϕ assuming all negotiation issues started at their earliest possible times. Given this assumption and a start time τ for a set of negotiation issues, the negotiation schedule generated from a negotiation ordering is unique.

Definition 2.6 *Given a start time* τ*, a negotiation ordering* ϕ *is* **valid** *if for every negotiation issue* v_i*, the finish time* $\varepsilon(v_i)_\phi$ *is no later than the negotiation deadline* $\epsilon(v_i)$*.*

The number of possible negotiation orderings for n negotiation issues $S(n)$ is: $S(n) = G(n) - I(n)$

$G(n) = 3^{\binom{n}{2}}$.

G(n) denotes the number of all possible different directed graphs based on n vertices. There are three possibilities: there is no edge between them; $(v_i, v_j) \in E$; $(v_j, v_i) \in E$. $I(n)$ denotes the number of graphs that contain cycles. $S(n) = O(3^{\frac{n^2}{2}})$

Definition 2.7 *A feature assignment* φ *is a mapping function that assigns a value* μ_{ij} *to every attribute* a_{ij} *that needs to be decided in this problem. For those attributes that already have been decided, value* μ_{ij} *is the decided value.*

Definition 2.8 *A feature assignment* φ *is* **valid** *if given the assigned values of those attributes, there exists at least one feasible local plan for all tasks and negotiation issues. It is assumed there is a function that can test if a feature assignment* φ *is valid; this function is domain dependent.*

Definition 2.9 *A negotiation solution* (ϕ, φ) *is a combination of a valid negotiation ordering* ϕ *and a valid feature assignment* φ*.*

The evaluation of a negotiation solution is based on the expected task-level successful values and decommitment penalties given all possible negotiation outcomes for every negotiation issue.

A negotiation issue has two possible outcomes: **successful** and **failure**.

Definition 2.10 *A negotiation outcome* χ *for a set of negotiation issues* $\{v_j\}, (j = 1, ..., n)$ *is a set of numbers* $\{o_j\}(j = 1, ..., n), o_j \in \{0, 1\}$*.* $o_j = 1$ *means* v_j *is successful,* $o_j = 0$ *means* v_j *fails. There are a total of* 2^n *different outcomes for* n *negotiation issues, denoted as* $\chi_1, \chi_2, ...\chi_{2^n}$*.*

Definition 2.11 *The expected value of a negotiation solution* (ϕ, φ) $\mathcal{EV}(\phi, \varphi)$ *is defined as:* $\mathcal{EV}(\phi, \varphi) = \sum_{i=1}^{2^n} P(\chi_i, \varphi) * (R(\chi_i, \varphi) + C(\chi_i, \phi, \varphi))$

$P(\chi_i, \varphi)$ *denotes the probability of the outcome* χ_i *given the feature assignment* φ*.*

$P(\chi_i, \varphi) = \prod_{j=1}^{n} p_{ij}(\varphi)$

$$p_{ij}(\varphi) = \begin{cases} p_s(v_j), (p_s(v_j) = \zeta_j(\varphi)) & \text{if } o_j \in \chi_i = 1 \\ 1 - p_s(v_j) & \text{if } o_j \in \chi_i = 0 \end{cases}$$

$R(\chi_i, \varphi)$ *denotes that the agent's utility increase given the outcome* χ_i *and the feature assignment* φ*.* $R(\chi_i, \varphi) = \sum_j \gamma(v_j) = \sum_j \theta_j(\varphi)$

v_j *is a root of a tree and* v_j *is* **task-level successful** $TS(v_j) = true$ *according to the outcome* χ_i*.*

$C(\chi_i, \phi, \varphi))$ *denotes the decommitment penalty according to the outcome* χ_i*, the negotiation ordering*ϕ *and the feature assignment* φ*.*

$C(\chi_i, \phi, \varphi)) = \sum_j \beta(v_j) = \sum_j \eta_j(\varphi)$

v_j *represents every negotiation issue that fulfills all the following conditions:*

1. v_j *is successful according to* χ_i*;*

2. *the root of the tree that v_j belongs to isn't* **task-level successful** *according to χ_i;*
3. *according to the negotiation ordering ϕ, there is no such issue v_k exists that fulfills all the following conditions:*
   (a) *v_k and v_j belong to the same tree;*
   (b) *v_k gets a failure outcome according to the outcome χ_i;*
   (c) *v_k makes it impossible for $root(v_j)$ to be* **task-level successful***;*
   (d) *the negotiation finish time of v_k ($\varepsilon(v_k)_\phi$) is no later than the negotiation start time of v_j ($\alpha(v_j)_\phi$) according to the negotiation ordering ϕ;*

2.3 Description of the algorithm

Based on above definition, we present a search algorithm that finds the best negotiation solution for a multi-linked negotiation problem $\mathcal{M} = (V, E)$.

Algorithm 2.1 *Finds the best negotiation solution.*
Input: $\mathcal{M} = (V, E)$, the start time for negotiation τ, a set of valid feature assignments $\{\varphi_k\}$, $k = 1, ..., m$.
Output:the best negotiation solution.
Generate all valid negotiation orderings $\{\phi_i\}$;
best_value = minimum_value;
best_ordering = null;
best_assignment = null;
for each negotiation ordering ϕ_i
  for each valid feature assignment φ_k
    if $\mathcal{EV}(\phi, \varphi) > best\_value$
      best_value = $\mathcal{EV}(\phi, \varphi)$;
      best_ordering = ϕ_i;
      best_assignment = φ_k;
return (best_ordering, best_assignment)

If the set of valid feature assignments is a complete set of all possible valid feature assignments, algorithm 2.1 is guaranteed to find the best negotiation solution. However, when the attributes have continuous value range, it is impossible to find all possible valid feature assignments. The following depth-first search (DFS) algorithm searches over the entire value space for all undecided attributes by pre-defined search step size and finds a set of valid feature assignments.

Algorithm 2.2 *Find a set of valid feature assignments.*
Input: $\mathcal{M} = (V, E)$; For each attribute a_{ij}, if a_{ij} is already decided, the value of a_{ij} is $defined\_value(a_{ij})$; if a_{ij} is undecided, the maximum possible range for a_{ij} is: $[min\_value(a_{ij}), max\_value(a_{ij})]$,
the search step size: $step_{ij}$.
Output: a set of valid feature assignments ω.
Generate the possible value set Ψ_{ij} for attribute a_{ij}:
If a_{ij} is already decided, $\Psi_{ij} = defined\_value(a_{ij})$;
Else $x = min\_value(a_{ij})$;
  Repeat
    add x to Ψ_{ij};
    $x = x + step_{ij}$;
    Until $x > max\_value(a_{ij})$
Generate all possible feature assignments φ_k based on the possible values in Ψ_{ij};
If valid(φ_k), add φ_k into ω;
Return ω;

Given the number of negotiation issues is n, and the number of valid feature assignments is m, the complexity of algorithm 2.1 is: $O(3^{\frac{n^2}{2}} * m * 2^n)$.

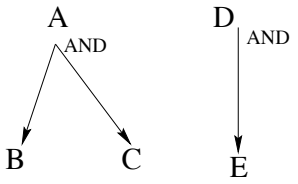

Figure 3: Interrelationships Among Negotiation Issues

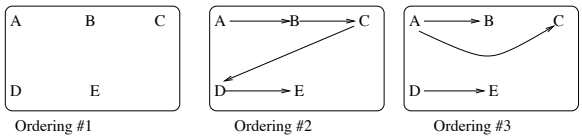

Figure 4: Three Possible Negotiation Ordering

3 Example

In this section, we demonstrate how the definition and the algorithm work on the supply chain examples in Figure 2. Figure 3 shows the relationships among the five issues for *Computer_Producer_Agent*: issue A is directly related to issue B and C, and issue D is directly related to issue E. Figure 4 shows three possible negotiation orderings for the five negotiation issues. Ordering #1 means all five issues are negotiated in parallel; ordering #2 means these five issues are negotiated one by one, first A, then B, then C, then D, and finally E; ordering #3 means that the negotiation on A is performed before the negotiation on B and C, the negotiation on D is performed before the negotiation on E. Suppose the negotiation start time $\tau = 0$, and the negotiation duration on each issue is the same $\delta(v_i) = 5$, then the following negotiation schedule is generated for negotiation ordering #3 according to the assumption that every negotiation issue starts at it earliest possible time:
$A[0, 5]B[5, 10]C[5, 10]D[0, 5]E[5, 10]$

Beside the attributes we presented in section 2.2, the following attributes need to be considered in this example:

1. time range (st, dl): the time range associated with an issue contains the start time (st) and the deadline (dl). If the issue is a task in negotiation, this task can only be performed during this range (st, dl) to have a valid result. The larger the time range is, the higher the probability of success this negotiation issue has, since it is easier for the other agent to arrange and schedule this task. For an incoming issue, this range is already determined by the other agent who proposes this request; for an outgoing issue, the agent needs to decide what time range to propose on this issue. The agent needs to make sure this time range is consistent with all other

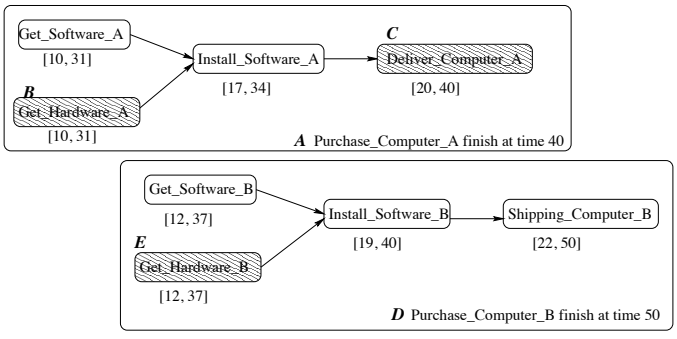

Figure 5: Partial Order Schedule

case#	Issue$v$	$p_s(A)$	ϵ	$e(A)$	β	Schedule	$dl-ft$	$e(A)*(dl-ft)$	$f(v)$	$p_s(v)$
#1	A	0.9	6	0.012	22.15	A[0-3]	0	0		0.9
	B		6		1.329	B[0-4]			3.0	0.8412
	C		6		1.329	C[0-4]			0.833	0.8829
#2	A	0.92	6	0.189	1.946	A[0-3]	21	3.964		0.92
	B		6		0.117	B[0-4]			1.0	0.7817
	C		6		0.117	C[0-4]			0.5	0.8766
#3	A	0.19	9	0.117	16.52	A[0-3]	0	0		0.19
	B		9		0.991	B[3-7]			3.0	0.8412
	C		9		0.991	C[3-7]			0.667	0.8799
#4	A	0.64	9	0.006	16.56	A[4-7]	0	0		0.64
	B		9		0.993	B[0-4]			2.428	0.8289
	C		9		0.993	C[0-4]			0.667	0.8799
#5	A	0.15	13	0.043	17.68	A[0-3]	0	0		0.15
	B		13		1.060	B[3-7]			2.428	0.8289
	C		13		1.060	C[7-11]			0.833	0.8829
#6	A	0.84	11	0.142	12.58	A[8-11]	9	1.278		0.84
	B		11		0.754	B[0-4]			1.428	0.7993
	C		11		0.754	C[4-8]			1.0	0.8859

Table 1: Examples of Negotiation Solution

time constraints of other issues, so it can find a feasible local schedule based on the commitment with this time range.

2. duration (d): if the issue is a task in negotiation, duration d is the process time requested to accomplish this task; if the issue is a resource in negotiation, duration d is the time needed for the usage of the resource.
3. flexibility (f): the flexibility is defined based on the time range and the duration: $f = \frac{dl-st-d}{d}$. The flexibility directly affects the **success probability**. For a negotiation issue with negative flexibility, the **success probability** is 0.
4. finish time (ft): the promised finish time for the task. For an incoming issue, the agent needs to decide the promised finish time (which must be no later than deadline dl the other agent requested) for this proposed task when it decides to accept this task.
5. regular reward (r): when an incoming task v is **task-level successful**, the agent's local utility increases by the amount of $r(v)$.
6. early reward rate (e): for a task in negotiation, if the contractee agent can finish the task earlier than the deadline request, it gets extra reward $e*(dl-ft)$.

The agent needs to find out how these features affect the **task-level successful reward** and the **success probability**. These relationships can be described as functions according to the agent's knowledge of each negotiation issue. In the negotiation process, for an incoming negotiation issue (such as A, D), the attribute needed to be decided is the promised finish time ft; for an outgoing negotiation issue (such as B, C, and E), the attributes needed to be decided are the start time (st) and the deadline (dl). The following functions describe how these attributes affect the **task-level successful reward** $\gamma(v)$ and the **success probability** $p_s(v)$.

For an incoming issue v, the **task-level successful reward** depends on the promised finish time ft: $\gamma(v) = r(v) + e(v) * (dl(v) - ft(v))$

For an outgoing issue v, the **success probability** depends on the flexibility $f(v)$, actually the time range $(st(v), dl(v))$: $p_s(v) = p_{bs}(v) * (2/\pi) * (\arctan(f(v)+c))$ [2]

[2]Obviously this function could be affected by the meta-level information from the other agent.

p_{bs} is the **basic success probability** of this issue v when the flexibility $f(v)$ is very large, c is a constant parameter used to adjust the relationship.

For every attribute that needs to be decided: start time (st), deadline (dl) and the promised finish time (ft), the agent can find it's maximum possible range using the partial order schedule as shown in Figure 5. Using algorithm 2.2, the agent can search over the entire possible value space, and using the partial order scheduler to test if a feature assignment is valid. A set of valid feature assignments is found and sent to algorithm 2.1 to find the best negotiation solution.

To make the output easier to understand, only issue A, B and C are considered in the following example. Table 1 shows the output of algorithm 2.1 on the example in Figure 3 given following parameters:

- regular reward $r(A) = 19$;
- negotiation duration $\delta(A) = 3$;
- negotiation duration $\delta(B) = 4$;
- negotiation duration $\delta(C) = 4$;
- $p_{bs}(B) = 0.95$;
- $p_s(B) = p_{bs}(B) * (2/\pi) * (\arctan(f(B) + 2.5))$;
- $p_{bs}(C) = 0.99$;
- $p_s(C) = p_{bs}(C) * (2/\pi) * (\arctan(f(C) + 5))$;

The different constant parameters for $p_s(B)$ and $p_s(C)$ specify that issue C has a higher **success probability** than B given the same flexibility.

The following parameters are randomly generated:

1. the **success probability** of A, $p_s(A)$;
2. negotiation deadline ϵ ($\epsilon(A) = \epsilon(B) = \epsilon(C)$);
3. early reward rate $e(A)$;
4. decommitment penalty $\beta(A)$, $\beta(B)$, and $\beta(C)$;

In both case 1 and case 2, the negotiation deadline $\epsilon = 6$, so the valid negotiation ordering has the three negotiation issues performed in parallel. In case 2 issue A has a higher earlier reward rate $e(A)$, and all issues have lower decommitment

Policy	Tasks Received	Task Accepted	Task Canceled	Decommit Penalty	Early Reward	Utility
Sequence Negotiation	60	60	37.25	73.82	0	358.09
Std.Dev.	0	0	2.6	11.8	0	57.4
Parallel Negotiation	60	60	23.70	333.20	29.06	385.20
Std.Dev.	0	0	2.6	47.6	17.0	86.8
Decision-Based Negotiation	60	60	25.78	56.65	185.79	779.16
Std.Dev.	0	0	2.4	23.5	47.8	62.3

Table 2: Comparison Of Performance

penalties β than in case 1, so the negotiation solution in case 2 arranges task A to finish 21 time units earlier than the requested deadline, and earns an extra reward of 3.964. In exchange, issue B and C have smaller flexibilities $f(B)$ and $f(C)$, hence lower success probabilities $p_s(B)$ and $p_s(C)$. In case 3 and case 4, the negotiation deadline $\epsilon = 9$. In case 3, issue A has a much lower success probabilities $p_s(A)$ than in case 4, so the negotiation on A is scheduled before the negotiation on B and C. In case 5 and case 6, the negotiation deadline $\epsilon = 11$ and the negotiation issues on A, B and C are sequenced according to the success probabilities; the issues with lower **success probability** start earlier. In case 6, issue A has a higher earlier reward rate $e(A)$, and all issues have lower decommitment penalties β than case 5, so the negotiation solution in case 6 arranges task A to finish 9 time units earlier than the requested deadline; this earns an extra of reward 1.278. In exchange, issue B and C have smaller flexibilities $f(B)$ and $f(C)$ and hence lower success probabilities $p_s(B)$ and $p_s(C)$. It is also important to notice that in all cases, issue B gets larger flexibility than issue C, but has a similar **success probability** to that of issue B. This occurs because it is much easier for issue C to achieve a successful negotiation according to the function that defines the relationship between the **success probability** and the flexibility.

4 Experiment

We have implemented an agent architecture including the agent controller, agent negotiation and execution components. The search and evaluation algorithms have been implemented so as to enable the reasoning in the multi-linked negotiation process as indicated in the examples described previously. We designed the following experiment to study how the different negotiation strategies which involve different reasoning efforts affect the agent's performance. The experimental environment was set up based on the scenario described in Section 3. New tasks were randomly generated with decommitment penalty $\beta \in [0, 25]$, early finish reward rate $e \in [0, 0.2]$, and deadline $dl \in [60, 70]$, and arrived at the contractee agents periodically.

In this experiment, *Computer_Producer_Agent* needs to deal with the multi-linked negotiation issues related to the incoming task *Purchase_Computer* and the outgoing task *Get_Hardware* and *Deliver_Computer*. The following three different negotiation strategies were tested:

1. Sequence Negotiation. The agent deals with the negotiation issues one by one, first the outgoing negotiation issues, then the incoming negotiation issues. The finish time promised is the same as the deadline requested from the other agent, and the outgoing issues get the largest possible flexibilities.
2. Parallel Negotiation. The agent deals with the negotiation issues in parallel. It arranges reasonable flexibility (1.5, in this experiment) for each outgoing task, and based on this arrangement, the finish time of the incoming task is decided and promised to the contractor agent.
3. Decision-Based Negotiation. The agent deals with the negotiation as the best negotiation solution suggests. The best negotiation solution comes from the decision-making process using the search algorithm 2.1.

The entire experiment contains 40 group experiments. Each group experiment has the system running for 1000 time clicks for three times and each time the *Computer_Producer_Agent* uses one of the three different strategies.

Table 2 shows the comparison of the *Computer_Producer_Agent*'s performance using different strategies. When the agent uses the sequence negotiation strategy, more tasks are canceled because of the missed negotiation deadline. When the agent uses the parallel negotiation strategy, the agent pays a higher decommitment penalty because the failure of the sub-contracted task prevents the incoming task to be **task-level successful**. The decision-based strategy is obviously better that the other two strategies[3]. It chooses to have the negotiation solution dynamically according to the negotiation deadline and other attributes. Under this experimental setup, it chooses the case where all negotiations performed in parallel about 13% of the time, it chooses the case where all negotiations performed sequentially about 38% of the time, and the other times it chooses the case where some negotiations are performed in parallel. This strategy enables the agent to get more early reward and pays fewer decommitment penalties.

The experiment shows that in a multi-linked negotiation situation, it is very important for the agent to reason about the relationship among different negotiation issues and make a reasonable decision on how to perform negotiation. This decreases the likelihood of the need for decommitment from previously settled issues and increases the likelihood of utility gain.

5 Conclusion and Future Directions

In this paper, we have presented a formal model of multi-linked negotiation problem, and defined how to evaluate a negotiation solution based on this model. We built a search algorithm that makes decisions on the negotiation ordering and the feature assignment for different attributes in negotiation. Experimental work shows this decision-making process leads to an improvement on the agent's performance. However, this search algorithm is a very simple complete search algorithm and its complexity makes it unrealistic for a real-time application with a large number of negotiation issues. In the future work, we would like to improve the search algorithm by introducing some heuristic search techniques to allow real time use of this methodology.

[3]Using t-test, with the 0.001 Alpha-level, the following hypothesis is accepted: when using decision-based strategy, *Computer_Producer_Agent* achieves an extra utility that is about 100% of the utility gained when using the sequence negotiation strategy, and 78% of the utility gained when using Parallel Negotiation Policy.

References

[1] Decker, K., Lesser, V. R. Quantitative Modeling of Complex Environments. In International Journal of Intelligent Systems in Accounting, Finance and Management. Special Issue on Mathematical and Computational Models and Characteristics of Agent Behavior., Volume 2, pp. 215-234, 1993.

[2] Sandholm, T. and Lesser, V. 1996. Advantages of a Leveled Commitment Contracting Protocol. Thirteenth National Conference on Artificial Intelligence (AAAI-96), pp. 126-133, Portland, OR, .

[3] Zhang, Xiaoqin and Lesser, Victor. Multi-Linked Negotiation in Multi-Agent System. Autonomous Agents And MultiAgent Systems 2002 (AAMAS 2002). To appear.